DETAILED DESCRIPTION

[Detailed Description of the Invention]
[0001]
[Field of the Invention]This invention relates to the programmed control device, the memory quota devices, and those methods of a computer.
[0002]
[Description of the Prior Art]First, the various definitions of term used in the specification of the invention in this application are shown below. The source sign is as follows.
[0003]The 4th edition (**********) of a JIS:JIS industrial use word great dictionary
31: Information technology term encyclopedia (Ohm-Sha **)
32: Information processing term large encyclopedia (Ohm-Sha **)
33: Tip software term encyclopedia (Ohm-Sha **)
3 A:SUPER ASCII Glossary Help on Internet (1) process (process) : Termini generales meaning processing or a process (3 2)
(2) Thread (thread) : the run unit which is the schedule target of a processor when the processing in which parallel execution is possible is divided into plurality in one execution environment called a process or a task, or the thing of flows of control. (3 3)
(3) Context (context) : (relating to an object-oriented system) The object which stores information required in order to perform a method is said. A context is materialized from the place which takes the information of the method object, program counter, and stack pointer containing the context of a call destination, and a program, an argument, and a temporary variable, and an evaluation stack. Although such an execution environment is taken as an object, it is called a heap language in the feature of the high-level language which supports a process etc. In another side PASCAL and ALGOL60, an execution environment is taken at a stack, and it is taken in FORTRAN in a fixed area. (3 1)
(4) task (task): -- one or more sequences (JIS) of the command dealt with by a control program in the environment of multiple programming or a multiprocessing as an element of the work which should be performed by a computer
(5) Garbage (garbage) : Object generated although not referred to from where, either. Garbage is collected by gar BEJIKO recta. (3 1)
(6) Garbage collection (Garbage Collection) : (relating to an object-oriented system) say the program which takes the lead in memory management. It is the method of all collecting the objects which will stop the main calculations if main memory is used up, move gar BEJIKO recta, and are not used any longer. If gar BEJIKO recta work by this method, since calculation stops, input and output will not react at all and it cannot use for a use to be answered [of real time]. (3 1)
(7) Interpreter (interpreter) : Translating program (JIS) which performs interpretation execution
(8) Real time (real time) : Term on the processing of data which a computer performs according to the requirements for time defined by external processing while it has a relation with processing of everything but the computer exterior. Real time. (JIS)
(9) Object (object) : it is an entity (substance) which combines the characteristic of pro cesura (defined procedure) and data, and calculation is performed by this and a

local state is stored. (3 1)
(10) Class (class) : Set of an object with the same procedure group and data structure. (3 3)
(11) Heap area (heap area) : workspace which is used if needed at the time of program execution. (3 2)
(12) : (scheduling) to schedule -- choose the job or task which should be dispatched. (JIS)
(13) Event (event) : Hardware/software should notify change of their own state to other hardware/software. Generally, by the notice in this case, the various parameters showing the kind of event or the state of hardware/software are summarized as a message, and it transmits to a partner. And in the side which received the notice of the event, suitable processing is performed from the parameter of a message, etc. (3A)
(14) Event flag (event flag) : the synchronous-communications mechanism between tasks which consists of a function for a task to wait generating of one or more phenomena, and a function which notifies the phenomenon. (3 3)
(15) Semaphore (semaphore) : structure for the system which processes two or more processes and tasks in parallel to perform the synchronization and message control between each process and between tasks, and interrupt processing. (3 3)
(16) Virtual machine (virtual machine) (virtual mathine) : environment where the application program independent of a specific platform built into two or more specific platforms (OS and hardware) is executed. If even the same virtual machine is provided, even if a platform changes, the same application program can be executed.
[0004](17) Java VM (Java Virtual Machine) : environment for executing a Java application program included in the operating system. A general program compiles a source code and takes the style in which the execution code optimized to each operating system is generated. Although the program written by Java is also created in the same procedure, the code after compile takes the form of the pseudo code independent of a specific operating system. This is loaded, it is a duty of a Java virtual machine which is performed changing into the code doubled with each operating system, and if even the same virtual machine is provided, the same code can be executed even if a platform changes.
[0005](18) Free area (free area) : the usable field on a heap area, free space.
[0006](19) Normal thread (normal thread) : Thread which performs processing as which real time nature is not required.
[0007](20) Mark table (mark table) : the table corresponding to 1 to 1 to the object which exists in order to investigate where an object is not referred to from. A mark is attached to the column of the table corresponding to the object when it is confirmed that a certain object has reference. When all the reference relation is investigated, since the object without a mark is unnecessary, it can be removed.
[0008](21) Life of an object (life time of the object) : time until an object is generated and eliminated.
[0009](22) Light barrier (write barrier) : Check change of the reference relation to an object, and when rewriting takes place, carry out a certain processing. In the case of this case, a mark is attached to the column of the mark table corresponding to the object which the reference to rewrite has pointed out when rewriting occurs.
[0010](23) Sweep (sweeping) : processing which removes the unnecessary object on a heap area.
[0011](24) Object generation (create the object) : generate a new object. Assign a part of heap area to an object, and, specifically, initialize the contents.
[0012](25) Object elimination (delete the object) : remove an unnecessary object.

Specifically, release the field secured in the heap area.
[0013](26) Reference (reference) : information which specifies the object B in order that a certain object A may access another specific object B. The pointer or index which specifically points out the object B.
[0014](27) Reference change (chenge the reference /reconnect the reference) : change reference into another object C from the present object B.
[0015]Now, when carrying out parallel processing of two or more threads on an operating system in the computer system of the conventional single processor, in order to carry out exclusive control using a shared memory, In order to take a synchronization among two or more tasks, a semaphore, an event flag, etc. are used conventionally and exclusive control is performed.
[0016]What is called dynamic memory control that allocates the memory of a single address space dynamically at the time of execution of a program without performing virtual memory for the purpose of operating a program under a little memory environment, for example etc. is performed conventionally. According to such dynamic memory control, if a memory area is not clearly released by a program, the memory area set and used no longer like real overshooting of a program occurs. As a result, the free areas which a program can use run short gradually. In order to avoid such a problem, it is made to perform processing which extracts the memory area (garbage) used no longer, collects these, and is made into a free area usable again (correcting) and which is called a garbage collection automatically.
[0017]Here shows to drawing 54 by making procedure of the conventional garbage collection into a flow chart. Although the algorithm of the garbage collection considers various methods, such as the mark & sweep method, the copying method, and the reference counting method, it illustrates about the mark & sweep method here. As shown in drawing 54, interruption is forbidden so that other threads other than a garbage collection thread may not be probably performed in the middle of a garbage collection, and it is set in single thread mode (s201). Then, the storage area (henceforth "a mark table") of the mark respectively corresponding to the object currently assigned to the field (henceforth a "heap area") which is the target of the garbage collection on a memory is cleared (s202). Then, the object currently referred to from which OBUJIEKU based on the information which shows the reference relation of the object currently assigned in the heap area is detected, and processing which attaches a mark to the position on the mark table corresponding to them is performed (s203). A mark will be attached to the mark table which the object currently referred to from neither of the objects is an object used no longer, and is equivalent to it by this processing. Therefore, the object which has not been marked is extracted as the field which can assign a new object, i.e., a free area, (s204). For example, this free area is generated as data of a list structure. Then, an interrupt inhibit is canceled and it returns to the multithread mode (s205).
[0018]Such a garbage collection is made to be started conventionally automatically, when the free areas of a memory decrease in number to the specified quantity.
[0019]When a garbage collection is performed, since the object of arbitrary sizes (memory size) is released arbitrarily, fragmentation occurs. Then, the memory compaction (only henceforth a "compaction") which packs the quota field of an object one by one from a head was performed so that the field where the continuous size is big could be taken.
[0020]
[Problem(s) to be Solved by the Invention]In the conventional system which has realized the function of the exclusive control mentioned above by using computer

resources, such as a semaphore and an event flag, when the resources are used by other threads, other threads must wait to release the resources. If the time of such resource waiting arises, in the real time nature system demanded, it will become a serious obstacle. That is, the thread in a resource waiting state cannot be processed, but the real time response of it becomes impossible until the resources are released.
[0021]For example, according to the method of the above-mentioned conventional garbage collection (henceforth "GC"). It takes time to find out the field which may be corrected as garbage, so that memory space is large, For example, 64-128 MB of heap area took for several seconds, and since GC was irregularly performed when free areas decreased in number to some extent, it was not able to use for the real time nature system demanded.
[0022]In the real time nature system demanded, if a certain event (interruption) occurs while performing a certain task, other threads according to the event will be processed, but. the time which the change of the thread takes -- as the worst value -- for example, tens -- to be less than microsec is demanded. However, once it cannot predict when GC is started as mentioned above, but it is started, CPU will become impossible [a meantime real-time operation] in order to concentrate on GC for several seconds.
[0023]An above-mentioned problem is a problem common to the system which not only the case of GC but a prolonged resource waiting state produces.
[0024]The purpose of this invention is to solve an above-mentioned problem by guaranteeing the exclusivity of processing, without using computer resources, such as a semaphore and an event flag, as a mechanism of a lock.
[0025]As the conventional copying method is incrementally performed as other methods of GC, the method of securing real time nature is shown in information processing Vol.35 No.11 pp1008 - pp1010. It is showy **** once to carry out parallel processing to other threads in time sharing, since discontinuation can be allowed in the middle of GC according to this method. However, in the method of being made to perform this copying method incrementally, since other threads cannot use a memory during a copy, only the thread of the pole part which does not use a memory can carry out parallel processing. In the copying method, since the memory area of a copy source and a copy destination must be secured, the utilization ratio of a memory is low and the system operated under a little memory environment is not turned to.
[0026]The method of giving a mark, whenever reference relation is changed, in order to correspond to a multithread by the mark & sweep method, The method for multi-CPU-izing which assigns processing to CPU only for GC is shown to information processing Vol.35 No.11 pp1006 - pp1008 by the name of "On-the-fly GC." However, when an object is always created and reference relation is changed in this method, The already marked tree was refollowed repeatedly, a new node had to be found and marked, and there was a problem of the case where mark work does not finish forever having arisen, or taking very long time. By this method, the system needed to be locked in the sweep.
[0027]Then, it is in other purposes of this invention making possible incremental GC which enabled it to complete GC certainly for a short time even if it made parallel processing of GC thread substantially possible with other threads and was interrupted at the arbitrary times of GC, though it was the mark & sweep method.
[0028]In order to cancel the fragmentation by GC, such as the conventional mark & sweep method, when performing a compaction, the great CPU power was needed. If this compaction was not performed, there was a problem that the utilization ratio of a memory fell.

[0029]It is in other purposes of this invention making a compaction unnecessary, and raising the utilization ratio of a memory.
[0030]In the conventional object-oriented system, Since the object with a long life which exists permanently, and the object with a short life extinguished for a short time were intermingled in the heap area, if GC is performed, it will be eliminated from the object of a short life, fragmentation will arise certainly, and a memory utilization ratio will fall suddenly. As for this, since it had to judge each time whether it was garbage also to the object which exists permanently when GC was performed, the CPU power was useless. On the other hand, although the method of the garbage collection according to what is called generation of considering that the object which was recognizing fixed time existence exists eternally, and removing the object from existence investigation is also considered from the former, There was a fault that an unnecessary object had to continue also by such a method, or continuation investigation of fixed time had to be conducted about each object.
[0031]Other purposes of this invention reduce fragmentation in consideration of the life for every object, and raise the utilization ratio of a memory, and it is in reducing the futility of a CPU power.
[0032]Here, the relation of the literature and the invention in this application which were discovered when a prior art search was performed is shown.
[0033](1) Incremental Garbage Collection of Concurrent Objects for Real-TimeApplication -- this paper to real-time GC in 1978 that Baker wrote, The rate of the processing time of required GC is searched for from the whole processing time. SUBJECT concerning the invention in this application is not solved.
[0034](2) By dividing finely the field which performs Distributed Garbage Collection for the Parallel Inference Machine:PIE64GC, It is the method of shortening each processing time and improving real time nature. It differs from the thing for all the fields fundamentally like the invention in this application. Scheduling is not described.
[0035](3) What developed the idea of Garbage Collection in Distributed EnviromentBaker, and was adapted for the network distributed environment. It is going to solve a problem peculiar to a network and differs from the invention in this application. It also becomes possible by combining the scheduler of the invention in this application with the view of this distributed environment to realize real-time more advanced CG by a distributed environment.
[0036](4) JP,1-220039,A
What controls the priority of the task started by the system call at the time of system call issue. There is only a common appearance in respect of priority control.
[0037](5) JP,3-231333,A
The size of an object is detected and securing a work area to a memory according to the size is shown once.
[0038](6) The concept of 586 to Institute of Electronics, Information and Communication Engineers Vol.80 No.6 pp592 multimedia operating system is shown.
[0039](7) Writing the size of an object to the top of the object on 12th time of Japan Society for Software Science and Technology D7-4 stack is shown. That is, there is relation only in that the size of an object is memorized.
[0040]A (8) 11TH Real-Time System Symposium "Incremental Garbage Collection ofConcurrent Object for Real-Time Applications" object. The view similar to the reference tree of a between is shown.
[0041](9) Performing memory assignment in relation to the life of U-38th time of Information Processing Society of Japan national conference 57 object is shown.

[0042](10) Lecture Notes in. GC is controlled according to a call of a Computer Science 259 "Garbage Collection in a Distributed Environment" application program interface, The technical thought about object reference is shown.
[0043]
[Means for Solving the Problem]An invention concerning claims 1, 18, and 25 answers a call of an application program interface which requests a start of write-in existence detection of data to the appointed memory area from a certain thread, When a flag which shows the write-in existence is set as a state without writing and there is writing of data to said appointed memory area, said flag is set as the state of writing in and being. And a call of an application program interface which requests an end of write-in existence detection of data to the appointed memory area from said thread is answered, and a value according to a state of said flag is returned to said thread.
[0044]Thereby, the exclusivity of processing is guaranteed, without using a computer resource as a mechanism of a lock. Namely, in the middle of processing of the thread A performed before a call of API which requests an end of write-in existence detection of data to the appointed memory area from a call of API which requests a start of write-in existence detection of data to the appointed memory area, It understands whether there was any writing to the appointed memory area by other threads by the thread A. If writing is not performed, the exclusivity of the appointed memory area is maintained. If writing is performed, processing of the meantime of the thread A is repealed, for example, exclusive control can be performed, maintaining a high response by a method of performing the processing again.
[0045]An invention concerning claims 2, 19, and 26 detects an object referred to from no object in a heap area of a memory, A thread which performs incrementally GC thread released as a free area which can memory assign [of other objects] a memory area of the object concerned is changed into a state higher than a priority of other threads, and a low state by turns.
[0046]Thus, by changing a priority of GC thread by turns in a state where a priority of GC thread is low. Priority is given to application by other threads, when there is no application which operates, GC thread is moved automatically, and a free area of a memory is expanded automatically. In a state where a priority of GC thread is high, although other threads are not performed, it is in a state where a priority is low, and when not carried out by other threads by GC continuing, a free area is prevented from being in an insufficient state chronically, and it can maintain always high performance.
[0047]An invention concerning claim 3 sets up time of a state where a priority of said garbage collection thread is high, by call of an application program interface. Time of a state where a priority is high at a side which calls API when thinking the response of a system as important, for example is shortened by this, Setting out of lengthening time of a state where a priority is high in thinking a throughput of the whole system as important is attained, and performance can be changed now according to needs of a system.
[0048]An invention concerning claim 4 sets up a cycle by state where a priority of said garbage collection thread is high, and a low state, by call of an application program interface. In being an application program which processes a continuous pulse thereby, for example and which needs CPU resources continuously, Overheads, such as context switching, can be made small by shortening a cycle and lengthening a cycle in an application program like the usual application program which exploits CPU resources intermittently.
[0049]An invention concerning claim 5 detects a temporal change of capacity of said

free area, and when capacity of said free area is a fall tendency, it lengthens time of a state where a priority of said garbage collection thread is high. This will adjust time of a state where a priority of a garbage collection is dynamically high, In a system and an application program which can avoid a situation of a free area becoming less insufficient with any application programs, and a free area can secure enough, a garbage collection can be suppressed to minimum.

[0050]An invention concerning claims 6, 20, and 27 performs a real-time thread according to generating of an event, When it is made to perform a non-real-time thread at the time of discontinuation of this real-time thread, or an end and free areas decrease in number to a predetermined value by execution of non-real-time threads other than GC thread, GC which is one of the non-real-time threads is performed.

[0051]Generally, the program as which a real time nature required thread and a thread which is not so exist in a real time nature system demanded, and real time nature is generally required can have little quantity of an object to generate, the quantity can be predicted, and it can design. On the other hand, prediction of quantity of an object generated is difficult for a real time nature program which is not demanded. Then, define quantity of an object expected that a real time nature program demanded generates, and a real time nature thread which is not demanded generates an object, When quantity of a free area of a memory decreases even near the quantity of an object which gave [above-mentioned] the definition, for example, scheduling is performed at the time and GC thread is performed. A free area is promptly secured by this and environment which can perform a real time nature thread demanded can always be maintained by it.

[0052]An invention concerning claims 7, 21, and 28 detects an object referred to from no object in a heap area of a memory, While releasing as a free area which can memory assign [of other objects] a memory area of the object concerned, based on quantity of said free area or a usage region of said object, it is made to perform two or more kinds of garbage collection threads from which a procedure differs selectively.

[0053]Generally, when performing GC, according to the algorithm, time etc. which power required of CPU, the amount of memory used, and GC take differ, respectively. Therefore, procedures of suitable GC differ according to quantity of a free area. As mentioned above, always efficient GC becomes possible by changing a procedure of GC according to quantity of a free area or a usage region.

[0054]An invention concerning claims 8, 22, and 29 detects distribution of size of an object assigned in a heap area of a memory, and makes an integral multiple of bigger fixed size than the center of a distribution quota size of the new object to said heap area.

[0055]Generally, in an object-oriented system, occurrence frequency distribution of a size (size on a memory) of an object generated at the time of execution of a program shows a normal distribution. On the other hand, when generating a new object, extract a field which can be assigned from a free area on a memory, but. By what bigger size than a center of a distribution of size of an object is made into quota size of a new object for as mentioned above. When a new object is assigned after the assigned object is eliminated, it can reuse, if it is an object smaller than the above-mentioned quota size. Since the above-mentioned quota size is bigger size than a center of a distribution of a normal distribution, many objects can reuse a memory area currently used previously. By this, even if it does not perform a compaction, a utilization ratio of a memory will improve. A CPU power for a compaction becomes unnecessary and a system with a high response can consist of small-scale CPUs.

[0056]After a means by which an invention concerning claim 9 detects distribution of

size of said object is built into a system at the time of the beginning of using of a system and detects distribution of size of said object, it shall be based on a program module separated from a system. A means by which an invention concerning claim 10 determines quota size of said object shall be based on a program module separated from a system, after being included in a system at the time of the beginning of using of a system and determining quota size of said object.

[0057]Thereby, once distribution of size of an object was detected, and after quota size of an object is determined, futility of a memory area and a CPU power is lost.

[0058]In a programmed control device with which an invention concerning claims 11 and 30 was provided with a means to generate an object, in a heap area of a memory, Inside of said heap area is beforehand divided into two or more sizes, and a field of the larger and smallest size than size of the object concerned is assigned to an object generate time.

[0059]An invention concerning claim 12 divides said heap area by call of an application program interface which makes an argument the number of partitions of two or more of said sizes, and each size.

[0060]In a programmed control device with which an invention concerning claims 13 and 31 was provided with a means to generate an object, in a heap area of a memory, An integral multiple of fixed size is determined as quota size of an object [/ in said heap area], and said fixed size is set up by call of an application program interface.

[0061]In a programmed control device with which an invention concerning claims 14 and 32 was provided with a means to generate an object, in a heap area of a memory, An integral multiple of fixed size shall be determined as quota size of an object [/ in said heap area]. Distribution of size of an object assigned in said heap area is set up by call of an application program interface, a call of this application program interface is answered and a bigger value than the center of said distribution is set up as said fixed size.

[0062]This measures distribution of size of an object assigned in a heap area of a memory with another device, a system, and an algorithm, Fixed size which becomes a basis of quota size of a determined object can be registered, and a utilization ratio of a CPU power and a memory can be raised.

[0063]When an invention concerning claims 15, 23, and 33 assigns an object generated by a class and this class as a template in a heap area of a memory, Data equivalent to time when an object was generated from said class is memorized, When detecting a life of the object concerned, providing data of this life in a class, when deleting said object, and generating an object from the class concerned, based on data of said life, a generation field of an object to said heap area is divided.

[0064]Generally, in an object-oriented system, since an object is generated by making a class into a template, an object generated from the same class has the life same in abbreviation. Then, when data equivalent to time when an object was generated from a certain class is memorized and the object is deleted, a life of the object concerned is detected, By generating an object to a different heap area based on life data, when memorizing this as life data of a class which generated the object and generating an object from the class, It is generated by field to which a long lasting object differed from an object of a short life. Thereby, fragmentation in a long lasting field falls substantially, and a utilization ratio of a memory improves. When performing GC, a CPU power consumed by GC can be reduced by processing preponderantly a field where an object of a short life is generated.

[0065]An invention concerning claims 16, 24, and 34 detects an object referred to from other objects in a heap area of a memory, and memorizes an existence state of

the reference concerned, Although released based on the memory content concerned as a field which can memory assign [of other objects] a memory area of an object referred to from no object, The 1st data of a tree structure that expresses reference relation of an object to a generate time of an object (this 1st data) For example, it is one variable which shows reference relation of an object and which is established in an object. It memorizes, The 2nd data in which an object of a portion into which reference relation of an object was changed is shown at the time of change of reference relation is memorized, An object which looks for the 1st data and is referred to is detected, while reading the 2nd data, it looks for the 1st data based on this data, and an object currently referred to is detected.

[0066]When performing a clearance of a sweep or a mark table by constituting in this way, Even if it does not stop other threads, and a new object is generated by other threads or reference relation between objects is changed into [clear] the inside of a sweep, or a mark table, When performing a sweep based on a mark table, a newly [the above] generated object is not eliminated. Therefore, since GC can be performed incrementally and discontinuation can be done freely, real time nature improves. It becomes possible to always operate GC thread in the background, and a utilization ratio of a memory can always be maintained highly. And time which mark grant by tree search takes can be shortened, inconvenience that mark grant is not forever completed by interruption can be prevented, and GC can be performed certainly.

[0067]An invention concerning claim 17 memorizes the object concerned of a reference destination as said 2nd data, only when said reference object detection means is the first object for a reference destination at the time of change of reference relation of said object to be detected by search of said 1st data.

[0068]Thereby, when a reference destination at the time of change of reference relation of an object is already marked, memorizing the 2nd data in piles is lost and time which the 1st search and mark of data take can be shortened that much.

[0069]

[Embodiment of the Invention]The composition of the programmed control device which is an embodiment of this invention, and a memory quota device is explained one by one with reference to drawing 1 - drawing 53.

[0070][A 1st embodiment] Drawing 1 is a block diagram showing the composition of the hardware of a device. A device consists of the memory 2 which memorizes a heap area, a mark table, etc. which generate CPU1 and an object group fundamentally, and I/O3 which perform input and output with the exterior. When it loads a required program to a memory from the outside, a program reads CD-ROM5 written in beforehand using the CD-ROM reading interface 4. This CD-ROM is equivalent to the program recording medium concerning the invention in this application.

[0071]Drawing 2 is a block diagram showing the composition of software. In the figure, a kernel portion manages CPU and a memory as resources, and realizes the function of the multithread by time sharing. VM (virtual machine) portion is software which performs the interface of a program and a kernel, and the whole hierarchy below VM makes it act as a Java virtual machine, seeing it from an application program here. (Java is a trademark of Sun Microsystems) A kernel and VM portion constitute JavaOS in this case. When a program is given to this VM by pseudo codes, such as a byte code, the interpreter which interprets this, the program module called according to that interpretation, etc. are included. The programs in the figure are the various threads by a pseudo code, and perform an internal program module via the above-mentioned interpreter.

[0072]Drawing 3 is a figure showing the reference relation of the object generated by

the heap area of a memory, and a relation with a stack. When an object is generated to a heap area, the reference relation to other objects [object / a certain] accomplishes the tree structure prolonged from a root node as shown in the figure. For example, if a global variable is declared, the object corresponding to the variable will be generated. The reference relation from the local variable on a stack as the stack which memorizes the information on an argument field, a return address, a local variable, workspace, etc. for every thread generated, for example, shown by a figure Nakaya seal to the global variable on a tree, etc. are memorized. These stacks are stored in the predetermined region outside a heap area.

[0073]Drawing 4 shows in detail the composition of the software shown in drawing 2 as a block diagram. In the figure, the GC module 10 is a program module of the various processing for performing GC, and the GC thread 11 performs GC by calling these modules. The interpreter 12 is this example, and when "object generation" of the GC module 10 is called when there is an object generation request from "the thread 4", and there is a change request of the reference relation between objects, it calls "mark grant" of the GC module 10.

[0074]In the example shown in drawing 4, although each thread is expressed with the pseudo code (byte code when [for example,] it is a Java applet), the compiler which changes a pseudo code into the native code to VM may be formed. (For example, in the case of Java, JIT (Just-In-Time compiler) etc. are provided.) In this case, since it is the thread described by the native code, direct access of the GC module 10 will be carried out, without passing the interpreter 12 shown in drawing 4.

[0075]By performing GC, drawing 5 is a figure showing the example which performed the compaction for canceling the fragmentation produced in a heap area. The hatching portion in a figure is an object, by carrying out the compaction of this, as shown in (B), fragmentation will be canceled and the continuous memory space will spread.

[0076]The program of the "compaction" of GC module shown in drawing 4 performs the above-mentioned compaction.

[0077]Drawing 6 is a figure showing the example of use of API for detecting context-switching generating existence. Processing 1 is performed, after publishing API#A which requests the detection start of context-switching generating existence before performing processing 1 as shown in (A) of the figure. API#B which requests the end of context-switching generating existence detection is published after the end of the processing 1. Since context switching has not occurred in the meantime in the example shown in (A), the next processing 2 is performed as it is. As shown in (B), supposing processing of thread #2 is performed in the middle of the processing 1 (i.e., if context switching has occurred), the processing 1 will be canceled after issue of API#B. For example, by processing which copies the contents of the field A of a memory to the field B, when context switching occurs during copy processing, the contents of the field A may change and the contents of the fields A and B may become inharmonious. Since it does not mean copying if inharmonious, the field B is repealed. This is the same as processing not having been performed from the beginning, and it is exactly having canceled the processing itself.

[0078]Drawing 7 shows the above-mentioned processing as a flow chart. First, processing 1 is performed after publishing API#A (s1->s2). After this processing 1 is completed, API#B is published and that return value is acquired (s3). When a return value expresses generating of context switching, the processing 1 is canceled and processing 1 is performed again (s4->s5->s1->s2). Processing is ended when a return value means un-generating of context switching. Thus, since the generating existence

of context switching within a predetermined period is known, if context switching occurs, it will become possible by canceling processing in the meantime and supposing that it is invalid to control exclusively, in spite of not locking the system actually.

[0079]Drawing 8 is a flow chart which shows the procedure in the kernel of above-mentioned API#A and API#B. If there is issue (system call) of API#A, the flag which shows the generating existence of context switching will be cleared. If API#B is published, it will return to a thread by making the state of the above-mentioned flag into a return value.

[0080]Drawing 9 is a flow chart which shows the procedure of context switching. If context switching is performed by the scheduler, context switching will be performed after setting the above-mentioned flag. That is, while storing the run state of the thread before a switch as a context, the context of the thread after a switch is read and it sets to the register of CPU, etc.

[0081]Drawing 10 is a flow chart which shows the procedure of the above-mentioned compaction. First, specify the object of the head in a heap area (s11), and the field for copying the object to the head of a heap area is secured (s12), As a certain data is not written in by other threads to the field during a copy, since, above-mentioned API#A is published (s13). And as shown in drawing 5, it puts by copying the object in a heap area to free space one by one from a head (s14). If the copy about one object finishes, above-mentioned API#B will be published (s15). It is obtained as a return value of API#B whether by this, context switching occurred in the period after publishing API#A until it publishes API#B. If context switching has occurred, the object which copied this time will be again copied to the already secured field (s16->s13-> ...). If context switching has not occurred, the following object is processed similarly (s16->s17->s18-> ...). Thus, it becomes possible to perform a compaction simultaneously with other threads moreover, without locking a system.

[0082]Although applied to the compaction in GC by the mark & sweep method in the above-mentioned example, when applying to GC by the copying method, processing shown in drawing 11 and drawing 12 is performed.

[0083]Drawing 11 is a flow chart of GC by the above-mentioned copying method. First, the object in the From field which moves a pointer to the root node of the data showing the reference relation of an object of a tree structure (s21), and is equivalent to the root node is copied to To field (s22). (A heap area is divided into a From field and To field, and reconstructs only the object which does not have garbage in To field by copying only the object in a From field which it should leave to To field.) Make the present From field into To field, and To field is made into a From field next time, The garbage collection by the copying method repeats the operation by turns. After that, a tree is followed, a pointer is moved to the following object in reference relation (s23), and the object is copied to To field (s24). This processing is performed about all the objects which can be followed by a tree.

[0084]Drawing 12 is a flow chart which shows the contents of the copy processing in drawing 11. First, the field for copying the object which should be copied to the prescribed position in To field is secured, After keeping other threads from a certain data being written in to the field during a copy and publishing above-mentioned API#A (s31), an object is copied to To field from a From field (s32). Then, above-mentioned API#B is published (s33). It is obtained as a return value of API#B whether by this, context switching occurred in the period after publishing API#A until it publishes API#B. If context switching has occurred, the object which copied this time will be again copied to the already secured field (s34->s31-> ...).

[0085]Drawing 13 and drawing 14 are flow charts which show the procedure in the case of detecting the generating existence of context switching, detecting whether it was rewritten before the object which is going to secure and copy exclusivity copying, and securing exclusivity.
[0086]When copying, as the field for copying the object which should be copied first to the prescribed position in To field is secured and a certain data is not written in by other threads to the field during a copy, since, API#C is published as shown in drawing 13 (s41). This API is an application program interface which requests detection of whether the receiving writing without the specified memory area occurred. Then, an object is copied to To field from a From field (s42). Then, above-mentioned API#D is published (s43). Whether after above-mentioned API#C is called before this API#D was called, the writing of a certain data generated this API in the appointed memory area is an application program interface returned as a return value. Therefore, if the memory area of the object which it is going to copy is specified, API#C is published and API#D also looks at a return value, it understands whether the object was referred to. Since the contents might write and it may have changed if the object is referred to, the copy of the object is redone (s44->s41-> ...).
[0087]Drawing 14 is a flow chart which shows the procedure in the kernel of above-mentioned API#C and API#D. If there is issue (system call) of API#C, a work area will be cleared to predetermined, and MMU (Memory Management Unit) will be set up so that an exception may occur, when the light of the field specified with the parameter of API#C is carried out. Issue of API#D will cancel the above-mentioned setting out to MMU so that an exception may not occur, when the light of the field specified with the above-mentioned parameter is carried out. And it returns to a thread by making into a return value the state of the flag set to the above-mentioned work region.
[0088]Drawing 15 is a flow chart which shows the contents of processing of MMU at the time of the above-mentioned exception generating. Generating of an exception will set the reference flag in the above-mentioned work region.
[0089]Although the above-mentioned example showed the case of GC by the copying method, it is applicable similarly [in the case of the compaction in the mark & sweep method].
[0090][A 2nd embodiment] Drawing 16 and drawing 17 are the figures showing the example at the time of changing automatically the priority of GC thread which can carry out a garbage collection incrementally.
[0091]As shown in (A) of drawing 16, high priority time raises the priority of GC thread (using a top priority), and low priority time performs by turns operation made low (for example, minimum priority).
[0092](B) of the figure shows the example at the time of performing simultaneously the thread of the priority of the degrees of middle other than GC thread. Namely, if the thread whose priority is higher than it will be in a Ready state when the priority of GC thread is low, If a context is switched and the priority of GC thread becomes high during execution of the thread, when a context will be switched to the GC thread and processing of the GC thread will be interrupted, threads other than the above-mentioned GC thread will be processed. Since GC thread will certainly be performed with a constant period if it does in this way, a free area is prevented from being in an insufficient state chronically, and it can maintain always high performance.
[0093]Drawing 17 is a flow chart which shows the procedure which the kernel about the change of the priority of a thread performs. The value of the priority enables it for issue of corresponding API to perform setting out of those with two or more steps,

and its value. Here, if API which sets up the value of a high priority is published when setting up two priorities of GC thread, as shown in (A) of drawing 17, the value will be set up as a value of the high priority of GC thread. If similarly API which sets up the value of a low priority is published, as shown in (B), the value will be set up as a value of the low priority of GC thread. If API which sets up the value and sets up the time of a low priority similarly is published as it is shown in (C) of the figure, when API which sets up the time of the high priority of GC thread is published, the value will be set up as shown in (D) of the figure.

[0094](E) of drawing 17 is a flow chart which shows the procedure to the scheduler of a kernel. Here, by an initial state, it assumes that it enters at queuing from which GC thread tends to receive assignment of resources at the cue of a high priority, and first, the data (data which identifies GC thread) is picked out from the cue of a high priority, and it inserts in the cue of a low priority (s51). And only the already set-up low priority time takes out the data which performs waiting for time (s52), and discriminates GC thread from the cue of a low priority continuously, and a scheduler inserts it in the cue of a high priority (s53). And only the high priority time when the scheduler is already set up performs waiting for time (s54). By repeating the above processing, as operation of the scheduler showed to drawing 16 (A), the priority of GC thread changes by turns. About threads other than GC thread, the same scheduling as usual is performed according to queuing of the cue corresponding to the priority of the thread, and as shown in (B) of drawing 16, context switching is made.

[0095]Drawing 18 shows the case where setting out of the high priority time of the above-mentioned GC thread is enabled by API. Drawing 19 is a flow chart which shows the procedure in the kernel according to the call of the API. If there is issue (system call) of this API, the above-mentioned high priority time will be set up with the parameter of that API (registration).

[0096]Drawing 20 is a flow chart which shows the example of the program using API whose setting out of the high priority time of the above-mentioned GC thread is enabled. First, the state of the 1st shown flag of a loop is seen (s61). Since it is an OFF state at first, this flag is turned on (s62), the present free area is investigated, and it is recorded (s63). The high priority time and low priority time of GC are the default value defined beforehand in the beginning. With a scheduler, after carrying out a fixed time stop (s64), shortly, If size comparison with the capacity of the free area investigated to last time and the capacity of the present free area is performed (s65) and the capacity of a free area is increasing, If high priority time of GC thread is shortened (s66->s67) and the capacity of a free area is decreasing, high priority time of GC thread will be lengthened (s68). The above processing is repeated. The priority of GC is dynamically adjusted automatically by this.

[0097]Drawing 21 shows the case where setting out of the cycle by the high priority time and low priority time of GC thread is enabled. When a cycle is long, (B) of (A) is when short.

[0098]Drawing 22 is a flow chart which shows the procedure in the kernel according to the call of periodic setting-out API whose setting out of the above-mentioned cycle is enabled. If there is issue of this periodic setting-out API, the value of the high priority time set up now and low priority time will be read, and that rate (ratio) will be calculated (s71->s72->s73). Then, high and low priority time are re-calculated according to the value of the cycle specified with the parameter of this periodic setting-out API (s74). And renewal of setting out of the high priority time and the low priority time is carried out (s75->s76).

[0099]For example, in being an application program which processes a continuous

pulse and which needs CPU resources continuously, Periodic setting-out API is published so that the cycle of GC thread may become short, and in the application program which exploits CPU resources intermittently like the usual application program, periodic setting-out API is published so that a cycle may become long.
[0100][A 3rd embodiment] Are the example which they perform GC thread when drawing 23 and drawing 24 are required, and secured real time nature, and in the example shown in drawing 23. The thread 1 and the thread 3 are a real time nature thread demanded and a real time nature thread (normal thread) as which the thread 2 is not required. The thread 3 is a GC thread. If the event 1 occurs in a normal state with many free areas of a memory when the thread 2 is performed, processing will be moved to the thread 1, and if processing of the thread 1 resulting from processing of the event 1 is completed, it will return to the thread 2. Similarly, if the event 2 occurs, processing will move to the thread 3. When a free area falls to the warning level defined beforehand by processing of the thread 2, processing of the thread 2 is interrupted and GC thread of the thread 3 is performed. If a free area is secured by this processing, it will return to processing of the thread 2. If the event 1 occurs in the middle of processing of GC thread, even if it is in the middle of GC, processing will move to the thread 1. Thus, in the real time nature thread demanded. Since there is little quantity of the object generally generated and a great portion of prediction is possible, If the above-mentioned warning level is set up according to it, inconvenience [say / that it becomes impossible to perform processing as which free areas decrease in number to the thread 2 substantially, and the real time nature of the thread 1 or the thread 3 is required of it by processing] is avoidable.
[0101]Drawing 24 is a flow chart which shows the procedure of the scheduler which processes the above-mentioned context switching. If the generating existence of the event was detected first (s81) and the event has occurred, a context will be switched to a corresponding real-time thread (s82). The event has not occurred, and if there are more free areas than a warning level, a context will be switched to the highest thing (at the example shown in drawing 23, it is the thread 2) of a priority among normal threads (s83->s84). If a free area becomes less than a warning level, a context will be switched to GC thread (s85).
[0102][A 4th embodiment] Drawing 25 shows the example of compulsive context switching in API which requests detection of context switching. Although it was made to understand whether context switching occurred after publishing API#A before publishing API#B in the example shown previously, When changing the priority of GC alternately with high and low, the example shown in this drawing 25 predicts that context switching occurs, while having published API for the above-mentioned exclusive control, and is kept from performing useless processing. Namely, if it judges whether required processing will complete API#A' by the time high priority time is completed and does not complete when API#A' is published while performing the high priority GC thread, the priority of GC thread is made low at the time. In the example shown in drawing 25, when the priority of GC thread became low, context switching will be carried out to the thread of an inside priority other than GC thread. Useless processing is avoided by this.
[0103]Drawing 26 is a flow chart which shows the procedure in the kernel according to the call of above-mentioned API#A'. If there is a call of this API, a context-switching flag will be cleared (s91) and the remaining time of the high priority time of GC thread will be acquired (s92). And size comparison with the exclusion time (time after publishing API#A' until it publishes API#B) and the above-mentioned remaining time which are the parameters of this API#A' is performed (s93). If the remaining

time is shorter than exclusion time, the priority of GC thread will be compulsorily made low (s94). The contents of processing and the contents of processing of context switching by API#B call are the same as that of what was shown in drawing 8 and drawing 9.

[0104][A 5th embodiment] Drawing 27 is a flow chart which shows the procedure for changing the algorithm of GC according to the amount of the memory (heap area) used. First, the threshold which shows which algorithm performs GC according to the value of memory usage and it is set up (s101), and memory usage is acquired (s102). This is the total value of the memory size of the object generated in the heap area. When memory usage exceeds threshold th1, GC is performed in the procedure of GC algorithm (1) (s103->s104). If memory usage exceeds threshold th2, GC will be performed in the procedure of GC algorithm (2) (s105->s106). It is the same as that of the following. Here, threshold th1 is smaller than threshold th2, and it performs processing which repeats by turns the height of the priority of GC thread shown in drawing 16 as a GC algorithm (1). As a GC algorithm performed when a threshold is high, irregular GC shown in drawing 23 is performed. GC itself is performed for example, by the mark & sweep method here. Usually, although a CPU load is light in the case of the former and GC is chiefly performed by this method, when a lot of memories are used for a short time by the normal thread, GC is preponderantly performed by the method shown in drawing 23. An always large free area is secured by this, and real time nature is maintained.

[0105][A 6th embodiment] Drawing 28 - drawing 30 are the figures about the processing for defining the quota size of the new object to the heap area of an object generate time. Distribution of the size of the object generally generated by execution of a program shows an abbreviated normal distribution, as shown in drawing 28. The center value is 32 and a grade settled among 64 bytes. Then, as shown in (A) of drawing 29, it is bigger size than this center value, and let size of the integral multiple of the exponentiation byte of 2 be the quota size of an object. Since only the quantity of the size of the object which should be generated was arbitrarily assigned as conventionally shown in (B) of the figure, the fragmentation of irregular size arises in the case of elimination of the object. According to the invention in this application, since the size of the object newly generated is an integral multiple of the above-mentioned fixed value, the reusability of a memory area increases and a memory utilization ratio increases in the whole. And depending on the case, a compaction becomes unnecessary.

[0106]Drawing 30 is a flow chart which shows the procedure of the object generate time. First, it asks for the distribution data of the occurrence frequency of the size of the object generated so far (s111). It is updated when having already asked for distribution data by last time. Then, it is the field as for which the memory which should be allocated this time is vacant, and a bigger field than the size of the object which it is going to generate is looked for, and an object is assigned to the memory area of the integral multiple of the above-mentioned fixed size (s112->s113->s114->s115). Although the exponentiation byte of the above 2 is a system constant, he does not necessarily have to make this value fixed size, and is arbitrary. If the case where a small object will be assigned to the field where size is big if fixed size is taken to too large a value will increase and fixed size is conversely taken to too small a value, the fields which are not recyclable to generation of an object will increase in number. What is necessary is just to determine that the utilization ratio of a memory will become the optimal, in determining the above-mentioned fixed size based on distribution data. If the value of the exponentiation of 2 is taken, for example even if it

is not an optimum value, the effect of becoming easy to do address determination will be done so.

[0107]Drawing 31 is a flow chart which shows the contents of processing of the program module which asks for the occurrence frequency distribution data of the size of the above-mentioned object, and the program module which determines the quota size of an object. First, the program module which asks for the occurrence frequency distribution data of the size of an object is loaded (s121), and the program module is started. That is, the size of each object generated (until starting and closing of an application program are performed 10 times, and until 24 hours passed, for example) is calculated for every size, and it asks for the distribution data until fixed time passes (s122->s123). Then, the distribution data is registered into a system (s124), and the program module which asks for the occurrence frequency distribution data of the size of an object is unloaded (s125). Then, the program module which determines fixed size is loaded (s126), and the program module is performed. That is, the distribution data in which the system was registered is acquired, it is bigger size than the center-of-a-distribution value, and the exponentiation byte of 2 is determined as fixed size, and the value is registered into a system (s127->s128). Then, the program module which determines fixed size is unloaded (s129).

[0108]Thus, once distribution of the size of an object was detected, and after fixed size is determined, the futility of a memory area and a CPU power is lost by unloading the program module for these.

[0109][A 7th embodiment] Drawing 49 shows the example which sets up the above-mentioned fixed size by API. As shown in a figure, in fixed size setting processing, fixed size setting-out API is called by making fixed size into an argument. In called API, it registers with a system by making an argument into fixed size. The size and fixed size of an object are compared with an object generate time (s211), if it is below fixed size, the free space of the fixed size on a heap will be looked for, and the found field will be assigned to an object (s212->s213). When exceeding fixed size, bigger free space than the object size on a heap is looked for, and the found field is assigned to an object (s214->s215).

[0110][An 8th embodiment] Drawing 50 shows other examples which set up the above-mentioned fixed size by API. As shown in drawing 50, in this example, first, object size-distribution setting-out API is called, and the distribution data of object size is set up. This distribution data performs and measures predetermined time predetermined application beforehand. Object size-distribution setting-out API answers a call, and sets an argument to an object size array variable. Then, it is bigger size than the center value, and the exponentiation byte of 2 is determined as fixed size, and this is set to a fixed size variable.

[0111][A 9th embodiment] Drawing 51 - drawing 53 are the figures showing the example which provides the quota size of the object in some sizes beforehand. Drawing 51 expresses the distribution of object size and the set of a region division which carried out predetermined time execution and measured predetermined application beforehand. When dividing a heap area beforehand, distribution of actual object size is measured, and division size and a number are defined so that it may become close to the distribution. For example, when a heap area is 2 MB, it is the set No.n byte k number m1 about the division size designated variable. 64 50002 256 100003 1k100004 4k50005 It is referred to as 32k500.

[0112]When setting up the above-mentioned division size, as shown in drawing 52, it carries out by calling division sizing API. As shown in drawing 53, called division sizing API sets an argument to the division size designated variable, and divides a

heap area according to it. That is, the loop counter n is first set to 0 (s221), and the size k and the number m are acquired from the n-th division size designated variable (s222). Then, the field of the size k is divided into m pieces from a heap area (securing by m pieces), and it registers with a list (s223). All the sizes are divided repeatedly, **********ing the loop counter n one time for this processing (s225->s222-> ...). When generating an object with the size of greater than 32 K bytes in the above-mentioned example, it assigns fields other than the field in a heap area divided [above-mentioned].

[0113][A 10th embodiment] Drawing 32 - drawing 34 are the figures showing the example for performing processing for raising the efficiency of GC in consideration of the life of an object. In the example shown in (A) of drawing 32, it divides into the field which generates the object of a short life, and the field which generates a long lasting object as a heap area, and a class is secured in a long lasting field. A class may be secured in other fixed areas. And the life flag which shows the life of the object generated in addition to the defining information as a template for generating an object is given to a class. This life flag is automatically generated to the generate time of a class. (B) of the figure is a figure showing the example of quota of the object to the conventional heap area.

[0114]Drawing 33 is a flow chart which shows the procedure of elimination of an object. As shown in the figure, when an object is eliminated, it judges whether the life of the object is long or short, and if short-life, the life flag of the class of the object will be set to a short life. For example, the time when the object was generated is stored in the field of an object, and the life of the object is searched for according to a difference with the time which eliminates the object. The above-mentioned time is good also considering the number of times of GC as a unit.

[0115]Drawing 34 is a flow chart which shows the procedure of generation of an object. If the life flag of the class shows a short life, an object is generated to a short-life field, that will be right and an object will be generated to ****** and a long lasting field.

[0116]Thus, by classifying the quota field of a memory according to the life of an object, in a long lasting field, fragmentation can be fallen substantially and the utilization ratio of a memory improves. The CPU power consumed by GC can be made small by performing GC preponderantly, for example about a life field.

[0117][An 11th embodiment] Next, incremental GC by the mark & sweep method is explained with reference to drawing 35 - 48.

[0118]Drawing 35 is a flow chart which shows the procedure of whole GC by the mark & sweep method. This GC repeats the clearance of a mark table, the mark grant by the above-mentioned tree search, and elimination (sweep) of an object, and performs them.

[0119]Drawing 36 is a flow chart which shows the contents of processing of "a mark clearance" in drawing 35. This processing once clears the contents of the mark table, moves a pointer to the head of a mark table first (s131), clears the mark of that position (s132), and moves a pointer to the position of the next mark (s133). This processing is repeated about all the marks (s134->s132-> ...).

[0120]Drawing 37 is a flow chart which shows the contents of processing of "elimination of an object" in drawing 35. First, a pointer is moved to the head of a mark table (s141), the existence of a mark is detected, if not marked, the position in the heap area of the object equivalent to the position on the mark table is calculated, and the object of relevance is eliminated (s142->s143->s144). Then, the pointer of a mark table is moved to the next and the same processing is repeated (s145->s146-

>s142-> ...). It leaves the object marked on the mark table by this, and other objects are eliminated from a heap area.
[0121]In order to explain easily, the mark grant in the mark & sweep method which will be the requisite first is explained.
[0122]Drawing 38 is a figure showing the procedure of the mark grant by tree search. As shown in (A), the reference relation expressed with a tree structure is followed from the root node 10 to each node, and a mark is given to the node (object) in reference relation. Specifically, the bit of the applicable position on a mark table is set. This tree structure is following a certain object's being constituted by the contents of the variable established in an object which show other objects of which being referred to, and following the reference relation of this object, for example, i.e., a tree.
[0123]As shown in (A), when mark grant is performed to the node number 3, an interrupt occurs, and by the interrupt processing. If it becomes a relation by which the object expressed with the node number 7 is referred to from the object which the reference relation to the object expressed with the node number 7 from a root node is broken off as shown in (B), and is expressed with the node number 2, Since the reference relation to the object expressed with the node number 7 from a root node is broken off when interrupt processing is completed, it returns to GC thread and mark grant is resumed, it will progress to the node number 8 which returns a pointer to a root node and is in the following reference relation as shown in (C). At this time, mark grant is not carried out to the node numbers 5 and 6. Then, it is necessary to give a mark about the object which follows a tree from the object and is referred to for the object with change of reference relation from the object.
[0124]Drawing 39 is a flow chart which shows the contents of processing of "generation of an object." Start the lock of a system to a kernel first (s151), and the vacant field in a heap area is looked for (s152), Required size is assigned to a bigger field than the size of the object which it is going to generate (s153->s154), mark grant (light barrier) of reference change is performed (s155), and a system is unlocked (s156).
[0125]Drawing 40 is a flow chart which shows the procedure of the above "mark grant of reference change." First, the position on a mark table is calculated from the object by which a reference change was made, and it is judged whether the mark of relevance is White. This White mark is expressed with 2 bits of 00, and means the state where it is not yet marked. Since it is already marked if it is not a White mark, processing is ended as it is. If it is a White mark, it will be marked on Gray. It means that this Gray mark is the object which was expressed with 2 bits of 01 and had reference change. When calculating the position of a mark from an object, it carries out by carrying out the address of an object 1/8, for example, and adding offset, or calculates with the consecutive numbers of an object.
[0126]Drawing 41 is a flow chart which shows the procedure of "mark grant by tree search." First, the pointer for following a tree is moved to the root node of a tree (s161), and the mark corresponding to the newly generated object is given (s162). Then, a tree is followed, a pointer is moved to the following object (s163), and this processing is repeated to the last of a tree (s164->s162-> ...). Then, a pointer is moved to the head of each thread stack (s165), and the mark corresponding to the object in a stack is given (s166). Then, a pointer is moved to the next of a stack (s167), and this processing is repeated to the last of a tree (s168->s166-> ...). Then, it moves to the next of a thread stack (s169), and the same processing is repeated to the last of a thread stack (s170->s166-> ...). Furthermore, processing about this thread stack is performed about all the thread stacks (s171->s172->s165-> ...). In this tree search of a

series of, when a Gray mark is detected even once on the way, search and mark grant are performed from a root node once again (s173->s161-> ...).
[0127]Drawing 42 is a flow chart which shows the procedure of "mark grant corresponding to an object" in drawing 41. This processing calculates the position on a mark table from the object currently generated, and gives a Black mark. This Black mark means the state where it is expressed and marked at 2 bits of 1x.
[0128]In the method of mark grant which was mentioned above, since a Gray mark will be given if interruption starts in the middle and the reference relation of an object changes, tree search must be repeated. It falls into the situation where GC is not performed forever, without never completing mark grant depending on the case.
[0129]Drawing 43 is a figure showing the procedure of "mark grant by tree search" for solving the above-mentioned problem. (A) shows a state when interruption starts in the state of (A) shown in drawing 38 and reference relation changes. Thus, when mark grant is performed to the node number 3, an interrupt occurs, and by the interrupt processing. The reference relation to the object expressed with the node number 7 from a root node is broken off, and a mark stack will be loaded with the data of a reference destination which expresses node number 7 if it becomes a relation by which the object expressed with the node number 7 is referred to from the object expressed with the node number 2. Then, since the reference relation to the object expressed with the node number 7 from a root node is broken off when interrupt processing is completed, it returns to GC thread and mark grant is resumed, the node number 8 which returns a pointer to a root node and is in the following reference relation as shown in (B) is marked. General tree search is ended at this time, and a mark is given about the object which follows a tree from the node shown by the contents of the mark stack as shown in (C) after that, and is in reference relation. As this shows (D), the mark grant about all the objects in reference relation is completed.
[0130]Drawing 44 is a flow chart which shows the procedure of "mark grant of reference change." Thus, the above-mentioned mark stack is loaded with the data in which the object by which a reference change was made is shown. Processing of generation of an object does not have what [was shown in drawing 39] strange straw.
[0131]Drawing 45 is a flow chart which shows the procedure of "mark grant by tree search." The portion of Steps s161-s172 is the same as Steps s161-s172 of the flow chart shown in drawing 41. After completing the tree search about all the thread stacks in the example shown in this drawing 45, From the object, pick out data from the Marks tuck (s181->s182), perform mark grant corresponding to the object shown by the data (s183), follow a tree, and to the last of a tree, Mark grant of the object in reference relation is performed (s184->s185->s183-> ...). It repeats, updating the pointer of a mark stack until a mark stack becomes empty about this processing (s181->s182-> ...).
[0132]Drawing 46 is a flow chart which shows the procedure of "mark grant corresponding to an object" in drawing 45. This processing calculates the position on a mark table from the object currently generated, and gives a mark.
[0133]Thus, by using a mark stack, the processing time of the whole which it becomes unnecessary to resume tree search from a root node, and mark grant takes can be shortened substantially.
[0134]Drawing 47 and drawing 48 are flow charts carried out as [abolish / still more useless processing time], when performing mark grant using the above-mentioned mark stack. Drawing 47 is a flow chart which shows the procedure of the above "mark grant of reference change." First, the position on a mark table is calculated from the object by which a reference change was made, and it is judged whether the

mark of relevance is White (s191->s192). This White mark means the state where it is not yet marked as mentioned above. Since it is already marked if it is not a White mark, processing is ended as it is. If it is a White mark, it will be marked on Gray (s193). It means that this Gray mark is an object with reference change as mentioned above. Then, a mark stack is loaded with the data in which the object of a reference destination is shown (s194).

[0135]Drawing 48 is a flow chart which shows the procedure of the above "mark grant corresponding to an object." This processing calculates the position on a mark table from the object currently generated, and gives a Black mark. This Black mark means the state where it is marked, as mentioned above. Processing of "generation of an object" is the same as that of what was shown in drawing 39.

[0136]Having given the mark, only when it was the first object for the object to be detected when a mark is given about an object with reference change Thus, a sake, The time which read-out of the time which the tree search by the contents of the Marks tuck takes, and a mark stack takes can be shortened.

[0137]A stack may not memorize the mark about an object with reference change, and it may use the buffer of FIFO.

[0138]Although the mark was given to the mark table in the embodiment, the information for a mark is provided in the inside of an object, and it may be made to give a mark directly to an object.

[0139]

[Effect of the Invention]According to the invention concerning claims 1, 18, and 25, the exclusivity of processing is guaranteed, without using a computer resource as a mechanism of a lock. Namely, in the middle of the processing of the thread A performed before the call of API which requests the end of write-in existence detection of the data to the appointed memory area from the call of API which requests the start of write-in existence detection of the data to the appointed memory area, It understands whether there was any writing to the appointed memory area by other threads by the thread A. If writing is not performed, the exclusivity of the appointed memory area is maintained. If writing is performed, processing of the meantime of the thread A is repealed, for example, exclusive control can be performed, maintaining a high response by the method of performing the processing again.

[0140]According to the invention concerning claims 2, 19, and 26, by changing the priority of GC thread by turns in the state where the priority of GC thread is low. Priority is given to the application by other threads, when there is no application which operates, GC thread is moved automatically, and the free area of a memory is expanded automatically. In the state where the priority of GC thread is high, although other threads are not performed, it is in the state where a priority is low, and when not carried out by other threads by GC continuing, a free area is prevented from being in an insufficient state chronically, and it can maintain always high performance.

[0141]Time of the state where according to the invention concerning claim 3 a priority is high at the side which calls API when thinking the response of a system as important, for example is shortened, Setting out of lengthening time of the state where a priority is high in thinking the throughput of the whole system as important is attained, and performance can be changed now according to the needs of a system.

[0142]In being an application program which processes a continuous pulse, for example and which needs CPU resources continuously according to the invention concerning claim 4, Overheads, such as context switching, can be made small by shortening a cycle and lengthening a cycle in an application program like the usual

application program which exploits CPU resources intermittently.
[0143]According to the invention concerning claim 5, the time of the state where the priority of a garbage collection is dynamically high will be adjusted, In the system and application program which can avoid the situation of a free area becoming less insufficient with any application programs, and a free area can secure enough, a garbage collection can be suppressed to minimum.
[0144]According to the invention concerning claims 6, 20, and 27, the real time nature thread which is not demanded generates an object, The quantity of the free area of a memory For example, since scheduling is performed at the time and GC thread is performed, when it decreases even near the quantity of the object which the real time nature thread demanded generates, A free area is secured promptly and the environment which can perform the real time nature thread demanded can always be maintained.
[0145]According to the invention concerning claims 7, 21, and 28, always efficient GC becomes possible by changing the procedure of GC according to the quantity of a free area or a usage region.
[0146]According to the invention concerning claims 8, 22, and 29, many objects can reuse the memory area currently used previously, and by this, even if it does not perform a compaction, the utilization ratio of a memory will improve. The CPU power for a compaction becomes unnecessary and a system with a high response can consist of small-scale CPUs.
[0147]According to the invention concerning claims 9 and 10, once distribution of the size of an object was detected, and after the quota size of an object is determined, the futility of a memory area and a CPU power is lost.
[0148]According to the invention concerning claims 11, 12, and 30, even if many objects can reuse the heap area of a memory without futility and do not perform a compaction, the utilization ratio of a memory improves. The CPU power for a compaction becomes unnecessary and a system with a high response can consist of small-scale CPUs.
[0149]According to the invention concerning claims 13, 14, 31, and 32, with another device, a system, and an algorithm. The fixed size which measures distribution of the size of the object assigned in the heap area of a memory, and becomes a basis of the quota size of the determined object can be registered, and the utilization ratio of a CPU power and a memory can be raised.
[0150]Since it is generated by the field to which the long lasting object differed from the object of a short life by generating an object to a different heap area based on life data according to the invention concerning claims 15, 23, and 33, The fragmentation in a long lasting field falls substantially, and the utilization ratio of a memory improves. When performing GC, the CPU power consumed by GC can be reduced by processing preponderantly the field where the object of a short life is generated.
[0151]When performing the clearance of a sweep or a mark table according to the invention concerning claims 16, 24, and 34, Even if it does not stop other threads, and a new object is generated by other threads or the reference relation between objects is changed into [clear] the inside of a sweep, or a mark table, When performing a sweep based on a mark table, the newly [the above] generated object is not eliminated. Therefore, since GC can be performed incrementally and discontinuation can be done freely, real time nature improves. It becomes possible to always operate GC thread in the background, and the utilization ratio of a memory can always be maintained highly. And the time which the mark grant by tree search takes can be shortened, the inconvenience that mark grant is not forever completed by interruption

can be prevented, and GC can be performed certainly.
[0152]According to the invention concerning claim 17, when the reference destination at the time of change of the reference relation of an object is already marked, memorizing the 2nd data in piles is lost and the time which the 1st search and mark of data take can be shortened that much.

[Translation done.]